Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Irt

Sistem obogatene resničnosti na osnovi detekcije obrazov

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Peter Peer
Asistent: as. Jernej Bule

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Sistem obogatene resničnosti na osnovi detekcije obrazov

Tematika naloge:

V diplomi zasnujte fleksibilno arhitekturo za zajem videa, detekcijo obrazov, obogatitev zajetega video toka ter prikaz tako obdelanega videa v realnem času. Obogatitev naj temelji na detekciji obraza in njegovih značilk, ki so izhodišče za zlivanje obogatitvenih objektov z resničnim svetom. Transformacije objektov (na primer stripovskih oblačkov, očal, pokrival ipd.) morajo delovati prepričljivo in brez opaznih zakasnitev. Ocenite kvaliteto detekcije značilk ter hitrost delovanja. Implementirajte tudi programski vmesnik funkcionalnosti diplomske naloge ter ga opišite.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Irt, z vpisno številko **63010044**, sem avtor diplomskega dela z naslovom:

Sistem obogatene resničnosti na osnovi detekcije obrazov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera in as. Jerneja Buleta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 22. september 2014 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Petru Peer in as. Jerneju Buletu za strokovno svetovanje, potrpežljivost in spodbudo pri nastajanju diplomskega dela.

Iskrena hvala moji družini za vso podporo in pomoč pri študiju ter vsem, ki ste mi vsa ta leta stali ob strani.

Kazalo

Seznam uporabljenih kratic

- API – programski vmesnik (angl. Application Programming Interface)
- BioID – anotirana baza obrazov
- CCD – naprava z napetostnimi spoji (angl. Charge–Coupled Device)
- CCV – knjižnica za računalniški vid
- CMOS – senzor z dopolnilnim kovinskooksidnim polprevodnikom (angl. Complementary Metal Oxide Semiconductor)
- GUI – grafični uporabniški vmesnik (angl. Graphical User Interface)
- IDE – integrirano programsko okolje (angl. Integrated Development Enviroment)
- OpenCV – odprtokodna knjižnica za računalniški vid (angl. Opensource Computer Vision Library)
- POSIT – poza iz ortografije in skaliranja z interacijami (angl. Pose from Orthography and Scaling with ITerations)
- SDK – programski paket za razvoj aplikacij (angl. Software Development Kit)
- SQL – strukturirani povpraševalni jezik za delo s podatkovnimi bazami (angl. Structured Query Language)
- XML – razširljiv označevalni jezik (angl. Extensible Markup Language)

Povzetek

V diplomski nalogi je predstavljeno programsko ogrodje kot temelj za aplikacije, ki omogočajo obogatitev resničnosti (angl. augmented reality) v realnem času. Programsko ogrodje temelji na detekciji obraza, ki zazna in sledi več obrazom naenkrat, hkrati pa poišče tudi obrazne značilke ter oceni položaj glave v prostoru glede na kamero. Delovanje ogrodja nato pokažemo na interaktivni marketinški aplikaciji, ki potrebuje ekran in kamero, tako da uporabnik lahko vidi sebe na ekranu. Aplikacija na obraz nariše določen objekt, na primer očala, brke, ipd. ter poleg obraza tudi stripovski oblaček s poljubnim tekstom.

Abstract

In this thesis we present software framework for real-time augmented realty applications. The framework is based on face detection and capable of tracking multiple faces at once. It detects facial features and estimates position of the face relative to camera. We implemented a practical example that uses framework in an application used for marketing. Application requires a screen and a camera, so that the users can see themselves on the screen. Application draws some predefined object on the user's face, for example glasses or mustache, and also a comicbook-like cloud near the face with chosen text.

Poglavje 1

Uvod

Človek zaznava z lahkoto tri-dimnenzionalno okolje. Če pogledamo fotografijo na kateri je skupina ljudi, lahko hitro preštejemo število ljudi na sliki, ugotovimo lahko tudi njihove lastnosti, kot so starost, spol, celo njihova čustva glede na izraz na obrazu.

Za računalnik je ta problem veliko težji kot se morda zdi, ker podatki iz senzorjev nikoli ne zajamejo vseh informacij iz okolja. Zaradi ne popolne informacije pri zajemu okolja večina algoritmov računalniškega vida temelji na verjetnosti in aproksimaciji. To pomeni, da se algoritem odloči za eno rešitev glede na verjetnost pravilnosti te rešitve.

V diplomski nalogi smo implementirali programsko ogrodje za detekcijo in sledenje obrazov v videu za namen realno časovne obogatene resničnosti. Ogrodje ima tri osnovne korake: zajem, interpretacija in prikaz podatkov. Vsi trije deli morajo biti procesirani v realnem času, ker je samo tako mogoče doseči interakcijo realnega okolja z dodatno računalniško generiranimi objekti [1].

V nadaljevanju tega poglavja razložimo kaj je obogatena resničnost ter naredimo kratek pregled uporabe obogatene resničnosti na različnih področjih.

Drugo poglavje je namenjeno opisu tehnologij, ki so potrebne za razvoj programskega ogrodja.

V tretjem poglavju opišemo aplikacijo, kot primer uporabe v tem po-

glavju opisanega programskega ogrodja. Aplikacija je namenjena za uporabo v marketingu. Potrebuje ekran in kamero, tako da se uporabniki vidijo na ekranu. Aplikacija na obraz vsakega uporabnika nariše poljuben element, na primer brke, očala, masko. Na koncu opišemo arhitekturo programskega ogrodja.

V četrtem poglavju smo testirali programsko ogrodje in sicer pravilnost določanja obraznih značilk na bazi BioID [12] ter hitrost delovanja.

V petem poglavju izpostavimo zaključke ter možne izboljšave.

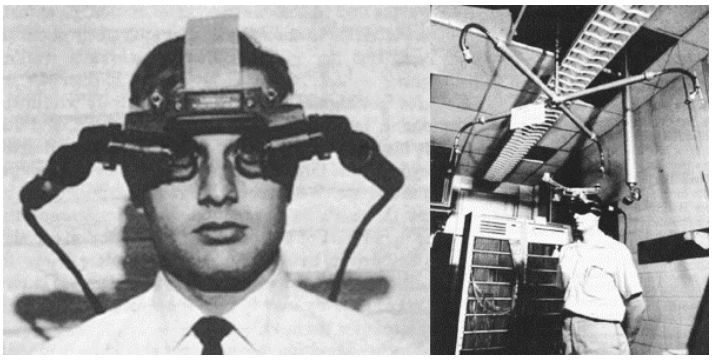
1.1 Obogatena resničnost

Obogateno resničnost (angl. augmented reality) lahko definiramo kot pogled na realni svet, ki je spremenjen oziroma obogaten z navidezno računalniško generirano informacijo. Cilj obogatene resničnosti je poenostavitev zaznave uporabnikovega okolja, tako da se prikaže navidezna informacija nad realnim okoljem [4]. Tako v nasprotju z virtualno resničnostjo (angl. virtual reality), kjer uporabnik zaznava samo sintetično okolje, obogatena resničnost doda navidezne objekte na realno okolje. Objekti vsebujejo informacijo, ki jo uporabnik sam ne more zaznati in izboljšuje uporabnikovo percepcijo njegovega okolja, zato je spekter uporabnosti obogatene resničnosti zelo širok. Obe obliki resničnosti sta del širšega pojma, ki se mu reče mešana realnost (angl. mixed reality) [19] (slika 1.1).

Slika 1.1: Mešana realnost.

1.2 Pregled uporabe obogatene resničnosti

Termin obogatena resničnost je bil formalno definiran v začetku 90-ih let prejšnjega stoletja, vendar pa njegovi zametki segajo še nekaj več kot dve desetletji nazaj. Leta 1968 je Ivan Sutherland naredil prvi sistem, ki si ga je uporabnik lahko namestil na glavo in s tem nad realnim svetom videl tudi računalniško generirane oblike, na primer preproste virtualne like, izrisane nad realnim zidom [24]. Seveda je bil Sutherlandov sistem okoren ter velik in zato ne prav zanimiv za množično uporabo (slika 1.2). V zadnjih letih pa je tehnologija s hitrim razvojem računalniške moči in s pomočjo ekonomske upravičenosti postala dostopna širši množici uporabnikov.



Slika 1.2: Prvi sistem obogatene resničnosti [24].

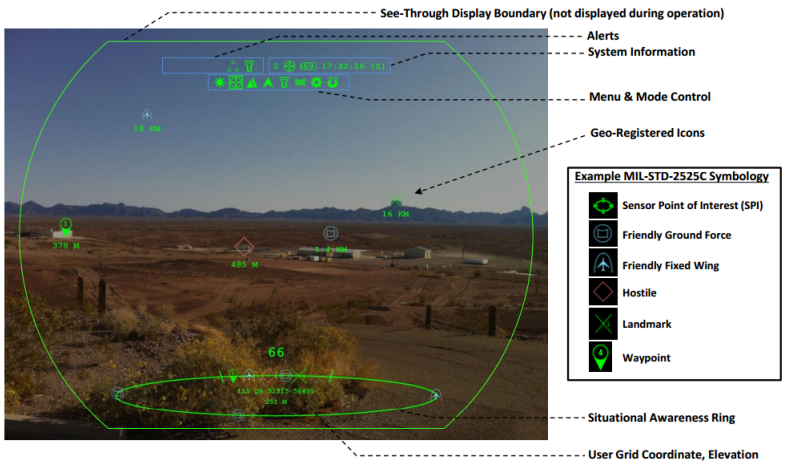
Medicina

Hiter razvoj medicinske tehnologije omogoča zdravnikom natančen vpogled v anatomske in druge podatke pacientov. Ker je zajem podatkov možen v realnem času, na primer snemanje s pomočjo ultrazvoka, potrebujejo zdravniki tudi nov način prikaza teh podatkov. S pomočjo obogatene realnosti lahko tako med operacijo popolnoma izkoristijo vse informacije, ki so jim na voljo [23] (slika 1.3).

Slika 1.3: Slika prikazuje, kako vidi kirurg glavne žile organa s pomočjo obogatene resničnosti [23], na drugi sliki pa vidimo uporabnost tehnologije pri diagnostiki pacientov [29].

Vojska

Vojska uporablja to tehnologijo že nekaj časa. Poslužujejo se transparentnih prikazovalnikov (angl. heads up displays), ki vojaku prikazujejo podatke v njegovem običajnem vidnem polju. V ameriškem podjetju Applied Research Associates so razvili sistem, kjer lahko vojak skozi vmesnik vidi tloris okolja, dodatno informacijo o poteku misije ali označi tarčo [22] (slika 1.4).



Slika 1.4: Applied Research Associates prikazovalnik [22].

Arhitektura in gradbeništvo

Poleg tega, da lahko arhitekt vidi predogled objekta na dejanski realni lokaciji, mu obogatena resničnost pomaga tudi pri projektiranju, saj lahko 2D načrt prikaže v treh dimenzijah. Na drugi strani lahko gradbeniki s sistemom vidijo celotno sestavo zgradbe, na primer položaj cevi, električne napeljave in podobno. Aplikacija CityViewAR [13] je bila razvita za mesto Christchurch iz Nove Zelandije, katerega je v letih 2010 in 2011 prizadelo več močnih potresov. Aplikacija s pomočjo geo lokacije na kraju podrtih zgradb prikaže slike teh objektov pred potresom (slika 1.5).

Slika 1.5: Aplikacija CityViewAR za obnovo podrtega mesta [13].

Izobraževanje

S pomočjo obogatene resničnosti se lahko učenci sami naučijo določenih nalog in veščin, saj teče interakcija z realnim in navideznim okoljem tako, da ni nobenih posledic v realnem okolju. Na primer gasilec se lahko nauči kako gasiti različne tipe požarov brez materialne škode, kirurg se lahko nauči izpeljati operativni poseg brez dejanskega posega v človeško telo [28].

Izdelava in popravila

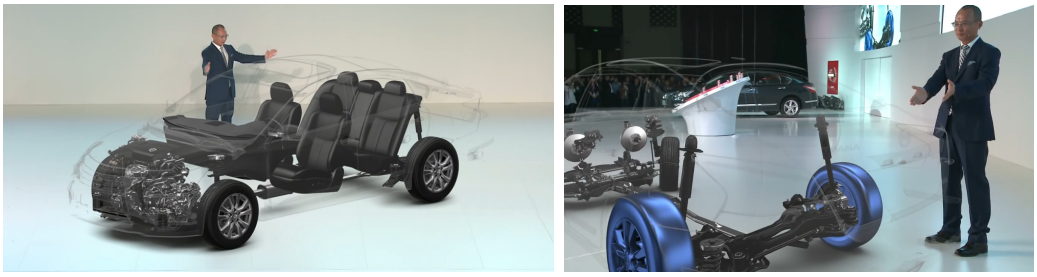
Obogatena resničnost se lahko uporablja tudi pri izdelavi in popravilu izdelkov, tako da uporabnik vidi 3D sestavo izdelka ter celo postopek izdelave ali popravila izdelka [10]. Podjetje Audi na primer za nekatere modele v letu 2015 pripravlja aplikacijo, ki bo nadomestila klasični priročnik za uporabo, s katero bodo lahko uporabniki interaktivno videli navodila za uporabo in servisiranje avtomobila kar z mobilnim telefonom (slika 1.6).

Slika 1.6: Audijeva aplikacija, ki nadomesti priročnik za uporabo [30].

Zabavna industrija in marketing

V zadnji letih se je tehnologija obogatene resničnosti močno razvila na področju zabavne industrije. Po nekateri projekcijah bo trg, ki je bil leta 2011 vreden 181,25 milijonov dolarjev, do 2016 zrasel na 5.155,92 milijonov dolarjev [31]. Zaradi takšnih napovedi lahko v naslednjih letih pričakujemo več sistemov, ki bodo za interakcijo uporabljali obogateno resničnost. Velike avtomobilske znamke že uporabljajo tehnologijo za predstavitev novih modelov avtomobilov, tako da lahko uporabnik preko ekrana na telefonu vidi 3D model avtomobila v okolju. Na sliki 1.7 lahko vidimo kako je Nissan uporabil obogateno resničnost za predstavitev novega modela Teana.

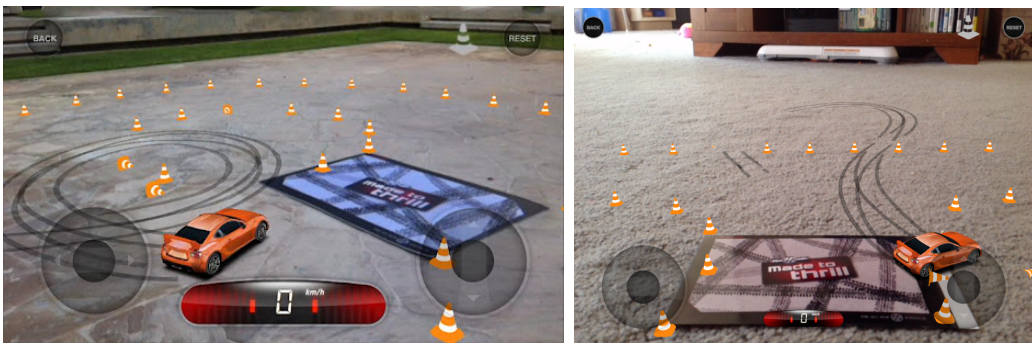
Tudi Toyota je z aplikacijo Toyota86AR uporabila koncept obogatene resničnosti za promocijo novega modela in sicer aplikacija projecira avto na ravno podlago, stranka pa ga lahko preko uporabniškega vmesnika vozi po



Slika 1.7: Nissanova predstavitev novega modela avtomobila [32].

površini (slika 1.8).

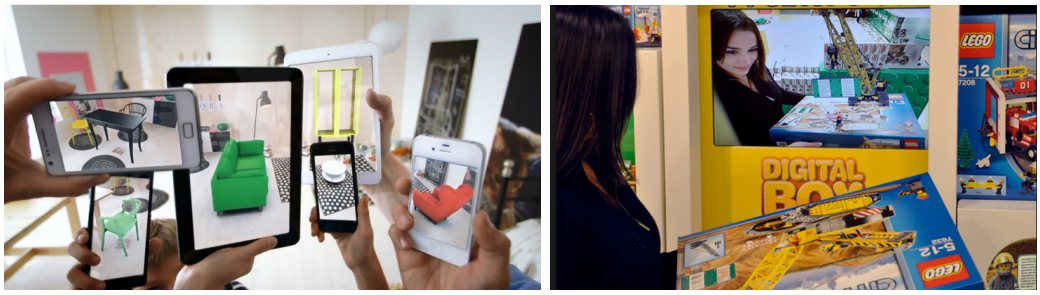
Pohištveni gigant Ikea je leta 2013 izdal aplikacijo, kjer lahko uporabnik postavi katerikoli kos pohištva v sobo iz kataloga, tako da na tla postavi tiskano izdajo kataloga. Podobno aplikacijo ima tudi Lego za 3D predogled modelov iz kock (slika 1.9).



Slika 1.8: Toyota86AR je aplikacija za promocijo novega Toyotinega modela [33].

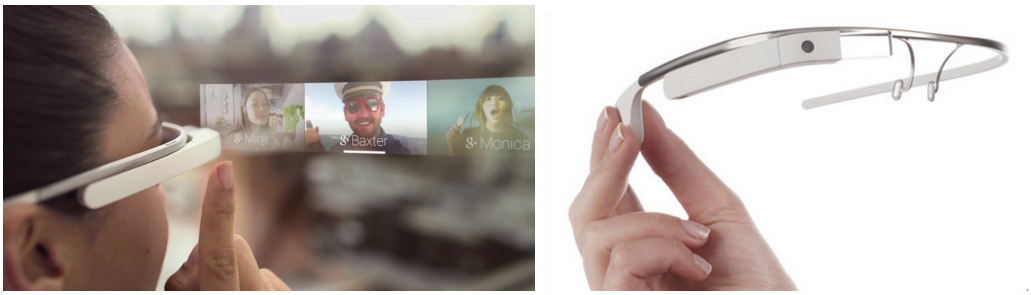
Takšen način uporabe obogatene resničnosti pa je omejen, saj mora uporabnik gledati v ekran, če želi videti dodatno informacijo. To oviro premagajo očala, ki projicirajo dodatno vsebino na leče. Z očali bo lahko uporabnik brez ovir opazoval okolico in pri tem imel obogateno informacijo o okolju, vendar pa takšna očala šele prihajajo na trg.

Na začetku leta 2014 je Google kot prvi začel prodajati takšna očala (slika 1.10). Pomembna novost, ki očala razlikuje od drugih tehnologij je ta, da jih lahko upravljamo z glasovnimi ukazi in s premikanjem glave brez



Slika 1.9: Aplikaciji Ikea [34] in Lego [35] za 3D predogled izdelov.

uporabe rok, saj imajo na desni strani mini prikazovalnik, kjer se projicira uporabniški vmesnik. Očala imajo v ohišju spravljeno strojno opremo in baterije, tehtajo pa samo 42 gramov, kar je impresiven dosežek. Upravljamo jih z vmesnikom na dotik na desni strani očal ali z besednimi ukazi. Na primer očala prižgemo z uporabo besedne zveze „Okay glasses". Imajo tudi kamero, s katero lahko slikamo ali snemamo. Ta funkcionalnost očal je zelo zanimiva, saj lahko uporabnik snema okolico iz svoje perspektive. Očala lahko povežemo s pametnim telefonom ali drugo napravo preko Bluetooth povezave.



Slika 1.10: Google Glasses [36].

Podjetju Google seveda sledi že veliko proizvajalcev kot na primer AtheerLabs [29], Space Glasses [37] in Telepathy [38].

Proizvajalci zagotavljajo, da bodo na voljo do konca leta 2014 ali vsaj na začetku leta 2015. Očala iz AthreeLabs gredo še korak dlje, saj sliko projicirajo na obe leči očal, tako uporabnik vidi 3D sliko generiranih objektov.

Poglavje 2

Uporabljena tehnologija

Naloga našega programskega ogrodja je iterpretacija podatkov iz kamere. Za to se uporabljajo rešitve s področja računalniškega vida. V tem poglavju so opisana orodja in algoritmi, ki smo jih uporabili.

2.1 Uporabljena orodja

Za implementacijo programskega ogrodja smo potrebovali orodja, ki so morala zadostiti naslednjim pogojem:

- potrebujemo integrirano razvijalsko ogrodje (angl. integrated development enviroment) za razvoj in razhroščevanje (angl. debuging) kode,
- programsko ogrodje mora biti sposobno procesiranja v realnem času,
- potrebujemo mehanizme za vzporedno procesiranje zaradi računske kompleksnosti algoritmov,
- potrebujemo programsko knjižnico, kjer so že implementirani vsaj osnovni algoritmi za računalniški vid,
- potrebujemo knjižnico za grafični vmesnik, ki mora vsebovati poleg grafičnih elementov tudi učinkovit mehanizem prikaza, saj potrebujejo računalniške vire algoritmi za računalniški vid.

Na podlagi naštetih pogojev smo za razvoj uporabili orodje Visual Studio 2012 in programski jezik C++.

2.1.1 OpenCV

Ker se področje računalniškega vida v zadnjih letih zelo hitro razvija, je temu primeren tudi razvoj knjižnic. Največja ovira pri implementaciji algoritmov je časovna in prostorska kompleksnost le-teh [2]. Zato mora knjižnica vsebovati tudi različne prijeme za hitro in vzporedno procesiranje. Najbolj znane odprtokodne knjižnice s področja računalniškega vida so: OpenCV [39], CCV [40], SimpleCV [41].

Knjižnica OpenCV velja za eno najboljših na tem področju. Leta 2013 je dobila prestižno Mark Everingham nagrado [42] za dosežke na področju računalniškega vida. Knjižnico so začeli razvijati leta 1999 v Intelu, leta 2008 je razvoj prevzela neprofitna organizacija OpenCV.org. Vsebuje več kot 2500 algoritmov iz vseh poddomen računalniškega vida. Ima vmesnike za programske jezike: C, C++, Phyton, Java in Matlab. Podprta je na operacijskih sistemih: Windows, Linux, Mac OS in Android.
Glavni moduli so:

- *Core:* definicija osnovnih podatkovnih struktur, ki jih uporabljajo ostali moduli.
- *Imgproc:* metode za procesiranje slike (geometrične transformacije, transformacije barvnega prostora, filtriranje, ipd).
- *Video:* algoritmi za procesiranje videa; sem spadajo algoritmi za oceno premikanja (angl. motion estimation), za sledenje objektom in metode za odstranitev ozadja iz videa.
- *Calib3D:* kalibracija kamere, algoritmi za stereo vid in za rekonstrukcijo 3D objektov iz slik.
- *Features2d:* metode za iskanje značilk in opisnikov ter metode za iskanje skupnih značilk iz več slik.

- *Objdetect:* metode za detekcijo in klasifikacijo objetkov kot na primer obrazov, ljudi, avtomobilov, ipd.
- *Highgui:* vsebuje elemente za grafični vmesnik in kodeke za video in slike.
- *GPU:* implementacija algoritmov, ki uporabljajo procesorsko moč grafične kartice.

Kot vidimo je knjižnica uporabna na vseh področjih računalniškega vida. Avgusta 2014 je izšla nova različica te knjižnice (različica 3.0), kjer so najpomembnejše novosti: dodatni moduli za lažjo uporabo algoritmov, boljša podpora za grafično procesiranje. Knjižnica ima tudi veliko pregledne dokumentacije in močno podporo skupnosti. Iz tega sledi, da je narejeno veliko dodatkov in razširitev. Mi smo pri svojem delu uporabljali različico 2.4.6.

2.1.2 Qt

Qt je med-platformska programska knjižnica, ki se predvsem uporablja za snovanje grafičnih vmesnikov. Qt je narejen v C++ in vsebuje generator kode imenovan Meta Object Compiler, ki zelo poenostavi programiranje. Ogrodje je na voljo kot odprtokodno ali s komercialno licenco. V namestitvenem paketu Qt-ja je poleg knjižnic tudi integrirano razvojno okolje, dokumentacija, primeri, izvorna koda in dodatki za različne platforme. Poleg modula za grafični vmesnik vsebuje tudi module za:

- delo s SQL in XML bazami,
- vzporedno procesiranje z nitmi (angl. threads),
- podporo za delo z omrežjem,
- enotni med-platformski programski vmesnik (API) za delo z datotekami.

Vse zgoraj opisane tehnologije zadostujejo pogojem, ki smo jih navedli na začetku poglavja, zato smo jih uporabili za razvoj programskega ogrodja.

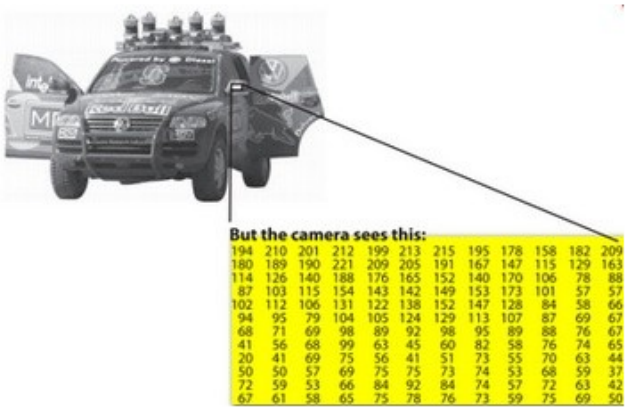
2.2 Uporabljeni algoritmi

2.2.1 Zajem slik na računalniku

Kamera vsebuje gosto tabelo neodvisnih senzorjev, ki spremenijo fotone svetlobe v slikovne točke, katere vrednost je intenziteta svetlobe. Število točk je od nekaj sto tisoč do nekaj milijonov, odvisno od kvalitete kamere. Kamere imajo ponavadi tri različne senzorje za vsako točko in sicer za rdečo, zeleno in modro barvo. Iz treh barv se rekonstruira katerakoli barva. Trenutno obstajata dve tehnologiji senzorjev: CCD in CMOS. Razlikujeta se po načinu zajetja fotonov svetlobe in prevedbe v digitalni signal. Prednost CCD tehnologije je boljša odpornost na šum, vendar pa je CMOS tehnologija veliko cenejša in porabi do 100-krat manj električne elergije [15].

Slika se shrani v obliki tabele, kjer vsaka vrednost definira pozicijo in intenziteto barve (slika 2.1). Digitalne kamere vsebujejo programsko opremo ki surove podatke iz senzorjev spremeni v datoteko, ki ima določen slikovni format, npr. jpg, gif, mpeg.

V programskem ogrodju smo uporabili razred `cv::VideoCapture` iz knjižnice OpenCV. Za zajem video posnetka smo uporabili funkcijo `open()`, ki odpre slikovni tok in funkcijo `grab()`, ki zajeto sliko iz toka vrne v formatu `cv::Mat`. Za zajem slike iz datoteke, smo uporabili funkcijo `cv::imread()`.



Slika 2.1: Primer digitalnega zapisa računalniške slike [39].

2.2.2 Algoritmi za detekcijo objektov na sliki

Kako računalnik iz dobljene tabele lahko določi in prepozna objekt? Osnova za prepoznavanje objektov je iskanje značilk (angl. feature detection). Značilke so zanimive točke na sliki, ki imajo določene lastnosti, ki jih definira oz. najde algoritem na sliki. Značilke so lahko robovi na sliki, kotni robovi ali homogena območja. Robustnost in natančnost teh algoritmov sta zelo pomembna, saj rezultat služi kot vhod v večino algoritmov, ki služijo za detekcijo in razpoznavo objektov. Seveda je vsaka slika objekta malo drugačna zaradi različnih faktorjev, kot so osvetlitev, različen kot slikanja, različna velikost in postavitev objekta. Zato želimo, da so značilke čim bolj invariantne na te faktorje. Ker naše procesiranje temelji na obrazih, smo potrebovali algoritme za lokalizacijo in detekcijo obraznih značilk.

2.2.3 Detekcija obraza

Naloga detektorja obraza je lokalizirati obraz in ugotoviti velikost obraza. Čeprav je digitalna slika samo množica svetlosti, je analiza obraza slike zelo zahtevna naloga.

Za to nalogo obstaja veliko različnih prijemov in vsak izmed njih ima svoje prednosti in slabosti. Nekateri algoritmi uporabljajo za osnovo barvo kože, obris obraza, obstajajo tudi bolj kompleksni algoritmi, ki uporabljajo za osnovo predloge (angl. templates), nevronske mreže ali različne filtre. Vsi ti algoritmi imajo preveliko časovno kompleksnost. Naše programsko ogrodje zahteva procesiranje v realnem času.

Algoritem, ki je robusten in natančen predvsem pa hiter, sta naredila Viola in Jones [26]. Osnova algoritma so Haarove značilke (angl. Haar-like features) [14].

V programskem ogrodju smo uporabili metodo `detectMultiScale()` iz razreda `cv::CascadeClassifier`, ki je implementiran v knjižnici OpenCV. Metoda potrebuje sliko v formatu `cv::Mat` in vrne seznam področji, kjer so bili detektirani obrazi, v formatu `vector<cv::Rect>`. Metoda omogoča

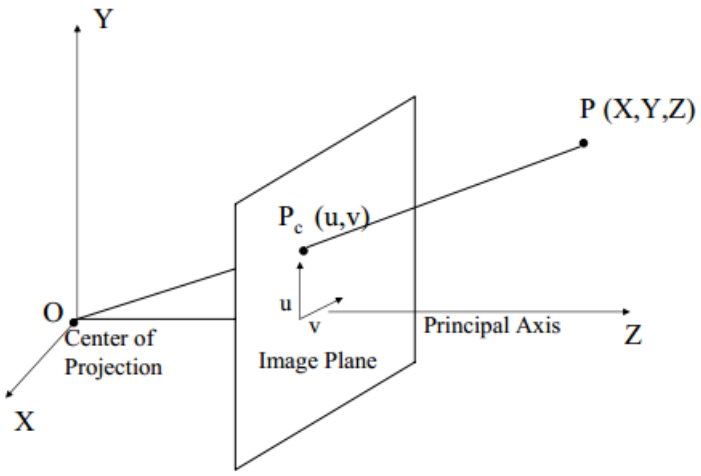
nastavitev dodatnih parametrov, ki pripomorejo k hitrosti ali robustnosti dektekcije, vendar smo se na podlagi testiranj odločili, da uporabimo privzete vrednosti.

2.2.4 Ocena oddaljenosti obraza od kamere

Ocena oddaljenosti objektov od kamere predstavlja velik problem, ker iščemo oddaljenost na dvodimenzionalni projekciji tri-dimenzionalnega prostora, kjer se izgubi informacija o globini objektov. Zato potrebujemo dodatno informacijo o realnem svetu.

Ena izmed rešitev je uporaba modela točkaste kamere (angl. pinhole camera model), kjer za oceno oddaljenosti potrebujemo poleg pozicije objekta na platnu še višino ali širino realnega objekta.

Slika 2.2 prikazuje kamero s centrom projekcije v O in glavno osjo Z. Platno slike (angl. image plane) je oddaljeno od centra za goriščno razdaljo f. Tri-dimenzionalna točka $P = (X, Y, Z)$ je preslikana na platno v koordinate $P_c = (u, v)$. Z uporabno podobnih trikotnikov vidimo, da so točke v relaciji po enačbi: $\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y}$.



Slika 2.2: Model kamere z luknjico.

Točko (u, v) lahko preberemo iz slike, goriščno razdaljo f pa je potrebno

oceniti s kalibracijo kamere. Tako za oceno oddaljenosti objekta Z potrebujemo še vrednosti za X in Y. Ker ocenjujemo oddaljenost obraza, lahko določimo X in Y vrednosti glede na povprečno velikost človeškega obraza. Seveda ne bomo dobili natančne vrednosti oddaljenosti obraza od kamere, ker so velikosti obrazov različne, vendar je ocena dovolj natančna za praktično uporabo [21].

2.2.5 Detekcija obraznih značilk

Detekcija obraznih značilk je osnoven korak za nalogi, kot sta poravnava in razpoznava obraza. Osvetlitev, poza obraza, obrazni izrazi, šum in slaba ločljivost obraza so glavni razlogi, ki otežujejo razpoznavo obraznih značilk [8].

Na tem področju je razvito veliko metod, večina uporablja za osnovo metodo učenja na podlagi videza (angl. apperence based) ali metodo geometričnih omejitev (angl. geometric constraints). Splošen pristop za iskanje obraznih značilk želi maksimizirati verjetnost obrazne značilke glede na sliko obraza, tako da je geometrična postavitev značilk anatomsko možna. Med najboljšimi metodami za reševanje problema sta pristopa aktivnega modela oblike (angl. Active Shape Model) in aktivnega modela videza (angl. Active Appereance Model).

Aktivni model oblike [5] je statistični model objekta katerega se iterativno deformira tako, da se najbolje prilega na dano sliko obraza. Ko algoritem konvergira, dobimo rezultat. Algoritem je dober za:

- iskanje objektov, ki imajo zelo dobro definirano obliko,
- klasifikacijo objektov po obliki,
- primere, kjer imamo reprezentativno množico primerov objekta,
- primere, kjer lahko s pred procesiranjem lokaliziramo objekt.

Iz tega vidimo, da je primeren za iskanje obraznih značilk, saj izpolnjujemo vse zgoraj naštete pogoje. Eden izmed slabosti algoritma je učenje baze, saj

morajo biti vključene vse možne deformacije obraza, ker drugače algoritem ne skonvergira.

Aktivni model videza uporablja za osnovo aktivni model oblike, ki ga nadgradi z dodatno informacijo. Ta je lahko tekstura ali osvetlitev objekta. Tekstura je uporabljena tako, da algoritem izmeri povprečno svetlost ali barvo v okolici točke, ki je definirana kot značilka obraza. Algoritem na vhodni sliki išče pozicijo, kjer je napaka med vhodno sliko in treniranim modelom najmanjša, zato ga lahko uvrstimo med nelinearne optimizacijske probleme [17].

V programskem ogrodju smo uporabili aktivni model videza, ki za dodatno informacijo uporablja algoritem SIFT [11]. Prednost teh značilk je v tem, da so invariantne na povečavo, rotacijo in osvetlitev [16]. Slabost tega pristopa je časovna kompleksnost algoritma in zato ni primeren za sisteme, ki delujejo v realnem času [9]. Zato je tak algoritem prepočasen za sledenje ampak zaradi robustnosti dober za detekcijo.

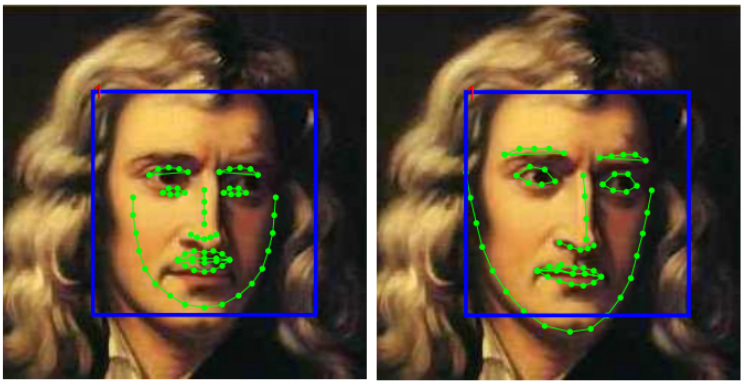
Takšen algoritem je implementiran v knjižnici Intraface z metodo `detect()`, ki za vhodne parametre potrebuje sliko v formatu `cv::Mat` in regijo detektiranega obraza v formatu `cv::Rect`. Algoritem vrne seznam 49 obraznih značilk in vrednost, ki pove zanesljivost detekcije.

2.2.6 Sledenje obraznim značilkam

Za sledenje potrebujemo algoritem, ki oceni obrazne značilke v realnem času. Uporabili smo metodo nadzorovanega spusta (angl. Supervised Descent Method) [27].

Algoritem iterativno poišče obrazne značilke iz začetne privzete pozicije z enačbo: $x_{k+1} = x_k + \delta x_k$, kjer je x_k pozicija obraznih značilk v k-ti iteraciji (slika 2.3). Cilj je priti iz začetne pozicije do nove pozicije tako, da se minimizira funkcija absolutne razlike SIFT opisnikov obraznih značilk (na primer opisniki za sredino nosu) in SIFT opisnikov, ki so kandidati za določeno obrazno značilko.

Algoritem je implementiran v Intraface knjižnici. Uporabili smo metodo



Slika 2.3: Na levi sliki vidimo začetno postavitev obraznih značilk. Desna slika predstavlja končno vrednost obraznih značilk [27].

`track()`, ki na vhodu potrebuje sliko v formatu `cv::Mat` in seznam že detektiranih obraznih značilk, prav tako v formatu `cv::Mat`. Metoda vrne seznam na novo detektiranih obraznih značilk ter zanesljivost detekcije teh značilk.

2.2.7 Ocena poze obraza

Ocena poze glave v prostoru lahko da veliko informacij o osebi in okolju, ker glavo uporabljamo tudi za neverbalno komunikacijo. Lahko ugotovimo na kateri objekt se oseba fokusira, odziv osebe na nek stimulans, tako da opazujemo hitro premikanje glave ipd.

Za reševanje tega problema obstaja veliko različnih pristopov [20]:

- Predloga izgleda (angl. apperance template methods): Ta metoda primerja novo sliko obraza s primeri obrazov, kjer je poza obraza vnaprej znana in poišče najboljši približek.

- Tabele detektorjev (angl. detector array methods): Ideja te metode je, da uporabimo detektorje obraza, ki so naučeni tako, da vsak najde samo obraz, ki je pozicioniran pod določenim kotom.

- Geometrične metode (angl. geometric methods): Uporabljajo lokacijo

zanimivih točk obraza kot na primer oči in ust, da določijo pozo obraza v prostoru.

- Metode s sledenjem (angl. tracking methods): Poizkušajo poiskati pozicijo obraza glede na razliko med dvema zaporednima slikama iz slikovnega toka.

- Hibridne metode: Združujejo zgoraj naštete metode.

Geometrična metoda POSIT [7] je hitri, iterativni algoritem, ki oceni položaj glave v prostoru glede na kamero, če imamo 2D koordinate točk projekcije objekta in 3D koordinate teh točk objekta.

Za definicijo 2D koordinat smo uporabili metodo nadzorovanega spusta, ki je opisana v prejšnjem poglavju. Za 3D koordinate modela glave se uporabi standardni model, saj je predpostavljeno, da so si geometrične lastnosti obrazov dovolj podobne, da lahko uporabimo ta model za ocenitev poze kateregakoli obraza.

Za oceno poze obraza smo uporabili metodo `EstimateHeadPose()`, ki je implementirana v knjižnici Intraface. Na vhodu potrebuje seznam obraznih značilk. Format seznama je isti, kot ga uprabljata metodi za detekcijo in sledenje obraznih značilk. Metoda vrne tri kote, v formatu `float`, ki definira pozo obraza v prostoru glede na kamero.

Poglavje 3

Razvoj aplikacije

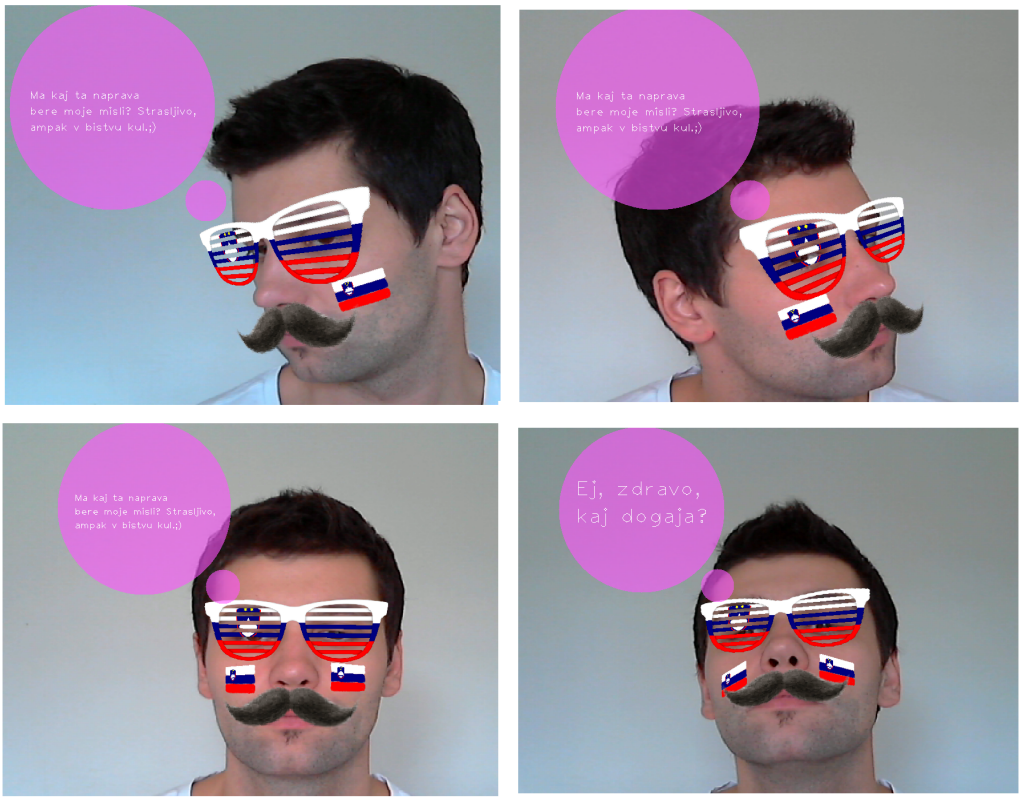
3.1 Cilji aplikacije

Programsko ogrodje služi kot temelj za aplikacijo. Namenjena je uporabi v marketingu in omogoča interaktivnost na osnovi detekcije obraza ter obogatene resničnosti. Aplikacija na obraz nariše poljubno sliko ter doda oblaček v katerem piše poljuben tekst. Zato je učinkovitost oglaševalske kampanje večja, saj si uporabniki tako bolje zapomnijo izkušnjo. Sistem ocenjuje oddaljenost mimoidočih, da lahko prilagodimo kdaj in kateri objekti se bodo narisali na obraz in vsebino teksta v oblačku nad obrazom (slika 3.1). Za vse te nastavitve smo razvili uporabniški vmesnik, ki omogoča hitro in lahko prilagajanje aplikacije glede na potrebe oglaševanja.

Slika 3.1: Primer uporabe aplikacije.

3.2 Pregled funkcionalnosti aplikacije

Osnovna funkcionalnost aplikacije je sledenje in risanje objektov na obraze mimoidočih. Zato je potrebna prepoznava in sledenje večim obrazom v realnem času. Aplikacija detektira pomembne značilke na obrazu, kot so oči, usta, nos, obrvi. Iz teh značilk lahko ocenimo velikost glave in pozicijo ostalih pomembnih obraznih značilk na primer čela, ušes, lic in brade. Oceni tudi pozo glave v prostoru glede na kamero. Ocena poze glave je zelo pomembna, saj le tako lahko določimo rotacijo in skaliranje objektov, ki jih narišemo na obraz (slika 3.2).



Slika 3.2: Primer skaliranja in rotacije objektov glede na pozo obraza.

V uporabniškem vmesniku nastavimo, katera slika se bo prilepila na določen del obraza. Pozicije na obrazu med katerimi lahko izbiramo so: center obraza, oko, obrv, lice, nos, usta, čelo. Možno je tudi premikanje

objektov po obrazu, tako lahko na primer nek objekt narišemo na desno stran levega očesa. Imamo tudi možnost skaliranja, tako lahko dosežemo poljubno velikost objekta na obrazu. Lahko določimo oblaček, v katerega napišemo poljuben tekst, oblaček pa se nato nariše nad obraz in izgleda, kot da aplikacija bere misli uporabnika (slika 3.2). Določimo lahko tudi interval oddaljenosti obraza od kamere. S temi nastavitvami dosežemo, da bo aplikacija glede na ocenitev parametrov naslikala objekte na obraz.

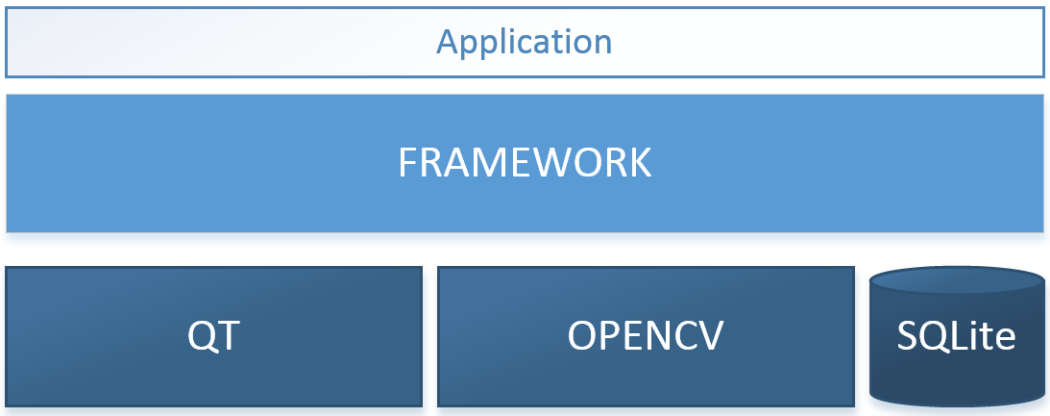
V grafičnem vmesniku lahko nastavimo ločljivost zajema slike, ločljivost slike za iskanje obrazov in ločljivost slike za sledenje obrazom. Slabša je ločljivost slike, manj procesorske moči porabimo in s temi nastavitvami lahko dosežemo procesiranje v realnem času tudi na manj zmogljivih računalnikih. Seveda je cena tega manjša natančnost ocene vseh parametrov. Nastavimo lahko maksimalno število obrazov, ki jim sledimo. Ta parameter je zelo pomemben, saj se računska kompleksnost primerno poveča glede na število obrazov. Če je število obrazov preveliko, se video vseeno prikazuje v realnem času, le pozicija in rotacija objektov malo zaostaja. Na primer, če je procesiranje možno vsak drug slikovni okvir, to pomeni, da se parametri rotacije in pozicije osvežujejo vsak drugi okvir in je efekt na videu, da objekti malo zaostajajo glede na premike obraza. Določimo lahko tudi točno določeno območje iskanja in sledenja obrazom na sliki. Glede na to, da lahko določimo interval oddaljenosti obrazov od kamere in določimo območje iskanja obrazov na sliki lahko točno določimo 3D prostor namenjen interakciji z uporabniki.

Ker aplikacija meri pozicijo glave v prostoru, lahko vizualiziramo tudi razmerje med časom sledenja obrazu in časom, ko je uporabnik gledal v ekran.

3.3 Implementacija programskega ogrodja

Programsko ogrodje smo implementirali s pomočjo tehnologij opisanih v drugem poglavju. Za osnovo smo uporabili knjižnici OpenCV in Qt (slika 3.3).

Ker je eden izmed pogojev za uporabo programskega ogrodja procesiranje v realnem času, smo izkoristili Qt podporo za paralelno procesiranje in dogodkovno vodeno (angl. even driven) komunikacijo med moduli. Za sledenje obrazu in detekcijo položaja obraznih značilk smo uporabili knjižnico Intraface [27], ki za osnovo prav tako uporablja OpenCV.



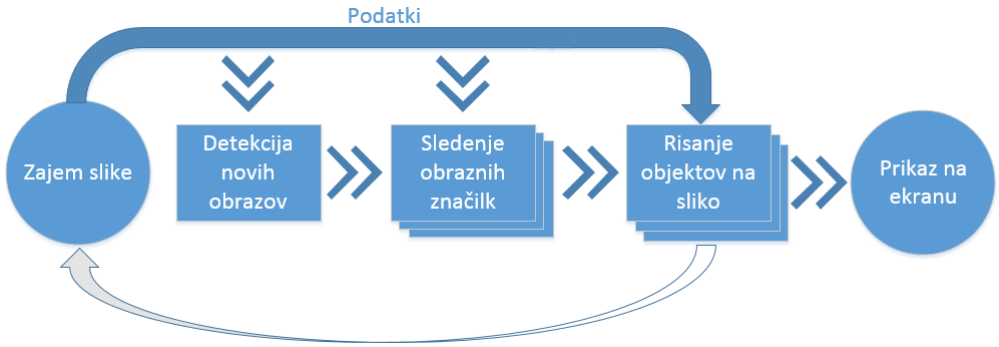
Slika 3.3: Arhitektura programskega ogrodja.

Programsko ogrodje smo implementirali v programskem jeziku C++ znotraj Visual Studia 2012 [47], ki nudi možnost razhroščevanja kode, optimizacijo kode in podporo za nadzor izvorne kode (angl. version control). Za nadzor smo uporabili orodje Git [43], ki omogoča kronološki pregled nad razvojem izvorne kode in je nujno orodje za kompleksne projekte, kjer sodeluje več ljudi.

Za shranjevanje parametrov programskega ogrodja smo uporabili označevalni jezik XML [44], saj omogoča shranjevanje parametrov v tekstovni obliki, tako lahko spreminjamo nastavitve v uporabnišem vmesniku in v tekstovnem urejevalniku. Za shranjevanje podatkov smo uporabili programsko knjižnico SQLite, ker omogoča shranjevanje podatkov brez potrebe po dodatni instalaciji SQL strežnika [45].

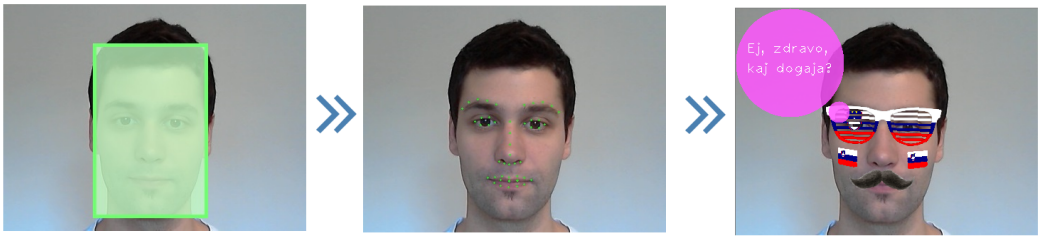
3.3.1 Opis delovanja programskega ogrodja

Programsko ogrodje smo razdelili na osnovne logične enote, kot kaže slika 3.4, kjer je predstavljeno osnovno zaporedje izvajanja. Definiramo lahko pet osnovnih modulov: zajem slike, detekcijo obrazov, sledenje obrazom, risanje objektov na obraze in prikaz na standardnem izhodu, navadno ekranu (slika 3.5).



Slika 3.4: Cevovod programskega ogrodja.

Zaradi pogoja, da mora programsko ogrodje procesirati v realnem času, je bilo potrebno implementirati vzporedno procesiranje vseh modulov. Zato je komunikacija med njimi asinhrona in poteka z uporabo signalov. Implementacijo takšne komunikacije omogoča knjižnica Qt.



Slika 3.5: Primer detekcije obraza, obraznih značilk in risanje objektov na obraz.

3.3.2 Potek glavne zanke

1. Zajem slike

   Ob zagonu programskega ogrodja se kreira nova nit, kjer se s signalom po določenem časovnem intervalu zajame sliko iz slikovnega toka. Nato se signalizira moduloma za detekcijo in sledenje, da je nova slika pripravljena.

2. Detekcija novih obrazov

   Modul za detekcijo skrbi za iskanje in lokalizacijo novih obrazov. Na vhodu potrebuje poleg nove slike tudi seznam lokacij obrazov, ki si že sledijo, tako da na tistem področju ne išče novega obraza. S tem rešimo problem večkratnega sledenja istemu obrazu in močno zmanjšamo čas detekcije, saj detektor porabi precej manj procesorske moči zaradi manjšega področja iskanja. Za detekcijo se uporablja algoritem Viola-Jones [26], ki je implementiran v knjižnici OpenCV.

3. Sledenje obraznim značilkam

   Modul za sledenje je implementiran tako, da procesiranje vsakega obraza teče vzporedno. Tako imamo toliko niti, kot imamo obrazov. Modul se zažene takrat, ko se je prejšnje izvajanje končalo in mu programsko ogrodje signalizira, da je pripravljena nova slika. Na vhodu poleg slike potrebuje tudi seznam obrazov, ki se že sledijo in seznam obrazov, ki so na novo detektirani. Nato modul poskuša poiskati novo pozicijo obraza. Če najde položaj obraznih značilk, signalizira programskemu ogrodju rezultate. Za sledenje obraznih značilk se uporablja metoda nadzorovanega spusta iz knjižnice Intraface [27].

4. Risanje objektov na sliko

   Ta modul živi v isti niti kot zajem slike, zato se izvaja zaporedno, takoj za zajemom nove slike. Ker lahko objekte rišemo na obraz vzporedno, je risanje implementirano z nitmi. Ko so vsi objekti narisani na trenutno sliko, modul signalizira programskemu ogrodju, da je slika

pripravljena za prikaz.

5. Prikaz na ekranu
   Prikaz slike je odvisen od aplikacije, ki uporablja programsko ogrodje. V našem primeru je grafični vmesnik implementiran s Qt knjižnico. Slika se iz OpenCV formata (`cv::Mat`) pretvori v Qt format za procesiranje na grafični kartici (`QPixMap`), tako je predvajanje slikovnega toka realizirano na grafičnem procesorju, kar razbremeni centralno procesorsko enoto in ostane več procesorske moči za ostale module.

3.3.3 Opis delovanja dodatnih modulov

V programskem ogrodju imamo poleg osnovnih modulov za procesiranje slike tudi pomožne module. V nadaljevanju bomo opisali njihovo funkcionalnost:

- Shranjevanje podatkov v bazo
  Ta modul je razdeljen na dva dela, eden skrbi za upravljanje z bazo XML, drugi del skrbi za upravljanje z bazo SQLite.

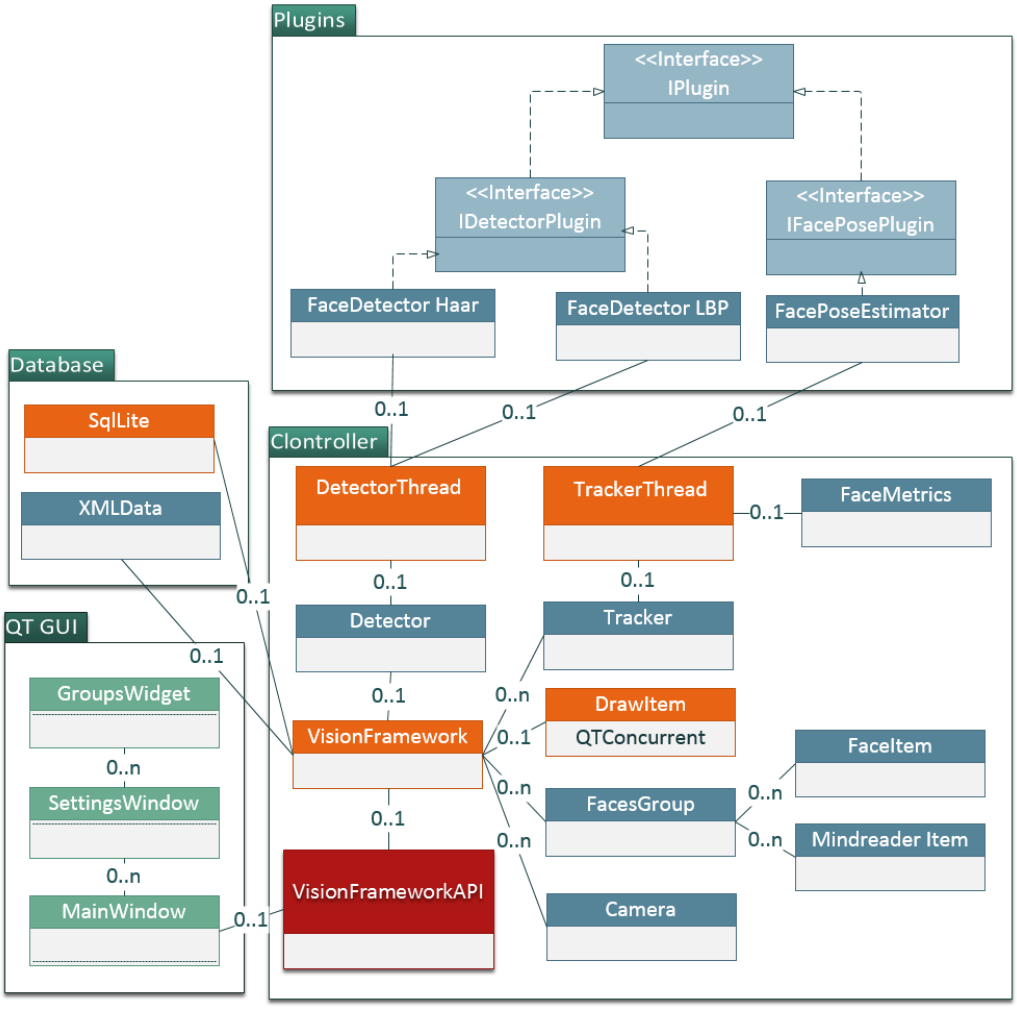
  Prvi del hrani podatke, ki jih lahko nastavimo v uporabniškem vmesniku opisanem v poglavju 3.2. Ti parametri se shranijo v tekstovni datoteki v formatu XML in se naložijo ob zagonu aplikacije ter shranijo ob zaprtju.

  Drugi del hrani podatke o obrazih v SQLite bazi. Implementiran je z nitjo, tako da lahko teče vzporedno s programskim ogrodjem. Ko programsko ogrodje izgubi obraz, shrani pomembne podatke v bazo. Ti podatki so: čas začetka sledenja, čas konca sledenja in odstotek časa, ko je uporabnik gledal v kamero.

- Vizualizacijo podatkov
  Modul je zadolžen za risanje grafov iz podatkov, ki so opisani v prejšnji točki. Implementiran je z razširitvijo Qt knjižnice z modulom QCustomPlot [46].

3.3.4 Struktura programskega ogrodja in aplikacije

Na sliki 3.6 je prikazan razredni diagram rešitve. Razredi, ki so obarvani z oranžno barvo, predstavljajo implementacijo s pomočjo nitk in tako omogočajo vzporedno procesiranje.



Slika 3.6: Razredni diagram rešitve.

V nadaljevanju so opisani najbolj pomembni razredi.

- `VisionFrameworkAPI`

  Vsebuje vse metode za uporabo programskega ogrodja. V dodatku je natačnen opis vseh metod razreda.

- `VisionFramework`
  Je glavni modul, kjer je implementirano osnovno zaporedje procesiranja.

- `DetectorThread`
  Skrbi za detekcijo novih obrazov.

- `Detector`
  Skrbi za upravljanje niti za razred `DetectorThread`.

- `TrackerThread`
  Funkcija tega razreda je sledenje obrazom. Za vsak obraz se naredi nova instanca razreda.

- `Tracker`
  Z upravljanjem niti skrbi za razred `TrackerThread`.

- `Camera`
  Skrbi za zajem videa ali slike iz kamere ali datoteke.

- `FacesGroup`
  Razred vsebuje podatke o tem, na katere skupine obrazov želimo narisati določene objekte. Za vsako skupino lahko določimo interval oddaljenosti od kamere in katere objekte, ki so definirani v razredu `FaceItem` in `MindreaderItem`, želimo narisati na obraz.

- `FaceItem`
  Hrani podatke o objektu, ki ga želimo narisati na obraz. Ti podatki so: slika objekta, pozicija na obrazu in podatek o velikosti objekta v primerjavi s širino obraza.

- `MindReaderItem`
  Vsebuje podatke in besedila za risanje oblačkov na obraz.

- `FaceMetrics`
  Razred vsebuje podatke o poziciji obraznih značilk in rotaciji obraza glede na kamero.

- `ItemPainter`
  Skrbi za risanje objektov na obraz, ki jih definira `FacesGroup` razred. Potrebne podatke o obrazu dobi iz razreda `FaceMetrics`.

- `dbSql`
  Razred, ki skrbi za upravljanje s podatkovno bazo SQLite. Glavna naloga razreda je shranjevanje podatkov v bazo za kasnejšo analizo.

- `XMLData`
  Razred skrbi za shranjevanje parametrov programskega ogrodja v XML formatu.

- `ChartsVision`
  Razširitev, ki vizualizira podatke iz baze z grafi.

- `Plugins`
  Je vmesnik (angl. interface), ki definira splošno strukturo dodatnih modulov (angl. plugins).

- `Qt GUI`
  V ta vmesnik spadajo razredi za grafični uporabniški vmesnik, kjer je prikazan video in grafični uporabniški vmesnik za nastavitve.

Poglavje 4

Testiranje programskega ogrodja

Testiranje smo razdelili na dva dela. V prvem delu smo testirali natančnost ocenitve obraznik značilk, ki je osnova za dobro obogatitev realnosti. V drugem delu smo testirali hitrost celotnega cevovoda.

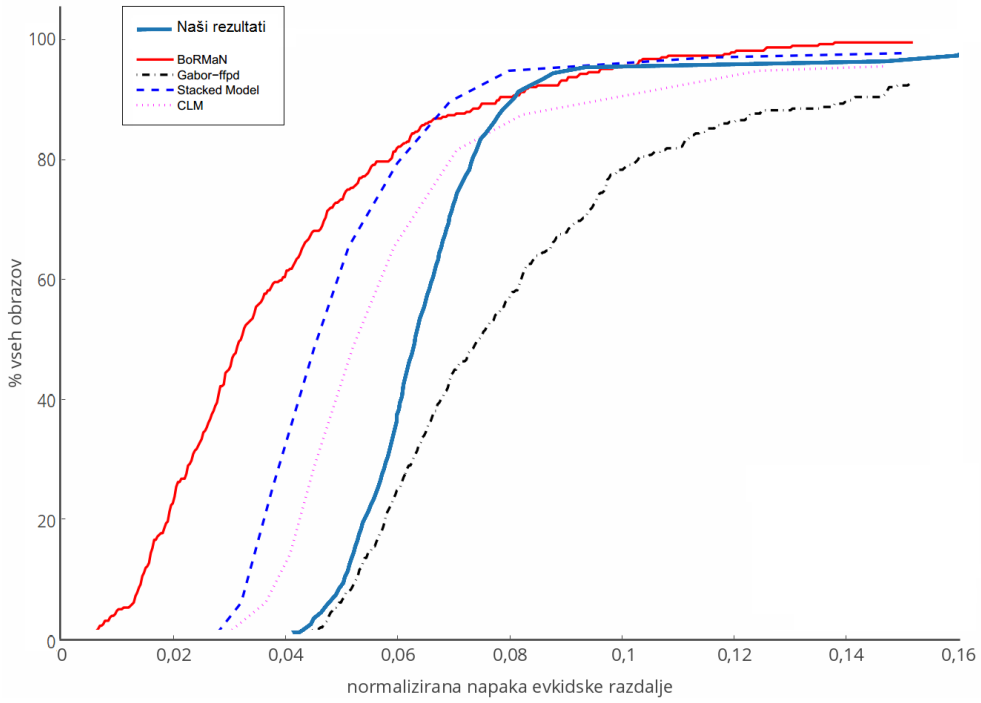
4.1 Testiranje natančnosti ocenitve značilk

Za testiranje obraznih značilk potrebujemo bazo, kjer so značilke ročno anotirane. Uporabili smo bazo BioID [12]. Vsebuje 1002 sliki ločljivosti 384×286 slikovnih točk. Natančnost ocene obraznih značilk bomo izračunali z enačbo (4.1). Takšno funkcijo za oceno natančnosti uporabljajo tudi ostali sistemi za iskanje obraznih značilk [3] [6] [18] [25]. V enačbi (4.1) je d_i evklidska razdalja med vsemi ročno nastavljenimi in detektiranimi značilkami, n je število obraznih značilk in s ročno nastavljena oddaljenost med očmi za normalizacijo. Rezultat je skalar, ki določi natančnost ocenitve obraznih značilk za vsak obraz.

$$m_e = \frac{1}{ns} \sum_{i=1}^{n} d_i \tag{4.1}$$

Na grafu 4.1 lahko vidimo odstotek vseh obrazov v odvisnosti od m_e. Za

90% obrazov je normalizirana napaka evklidske razdalje med točkami manjša kot 0.08, kar se lahko primerja z najboljšimi algoritmi do leta 2011, ki so bili testirani na tej bazi [3]. Manjšo napako kot 0.05 pa ima samo 10 % obrazov, kar je dosti slabše od primerjanih algoritmov (pri algoritmu [3] ima več kot 80% obrazov manjšo napako), vendar je treba upoštevati dejstvo, da je bil naš algoritem naučen na drugi bazi. Poleg tega tudi ročno nastavljene vrednosti niso povsem natančne in to prispeva k napaki.



Slika 4.1: Na sliki vidimo primerjavo naših rezultatov testiranja s sistemi: [3] [6] [18] [25].

Iz rezultatov v tabelah 4.1, 4.2 in 4.3 vidimo, da pri večini obrazov metoda najde vse obrazne značilke. Napaka je največja za zunanji in notranji rob obeh obrvi, najboljše pa algoritem oceni obe očesi ter zunanja roba ustnic.

% vseh obrazov	Levo oko	Desno oko	Levi rob ustnic	Desni rob ustnic	Zgornji rob ustnic	Spodnji rob ustnic	Nos
10	0,0138864	0,0139047	0,0248676	0,0261796	0,022825	0,0136832	0,0178871
20	0,0183864	0,0203518	0,0313218	0,0338744	0,0307862	0,0184496	0,0254364
30	0,0222858	0,0252734	0,0359871	0,0406568	0,0373948	0,0236293	0,0317645
40	0,0259806	0,0291972	0,0402346	0,0461684	0,0434232	0,0271812	0,0397418
50	0,029327	0,0346395	0,045324	0,052658	0,051044	0,031343	0,049053
60	0,033462	0,038775	0,0512176	0,0582882	0,058545	0,0359938	0,0589256
70	0,0378204	0,0435824	0,05673	0,0654167	0,0681805	0,0404075	0,069049
80	0,0441086	0,0521514	0,0625982	0,0761222	0,0825288	0,0468868	0,0885268
90	0,0573428	0,0681529	0,0724872	0,093675	0,1068305	0,0624233	0,1145163
100	4,209642	3,871386	4,129804	3,885626	3,91083	4,023895	3,942653

Tabela 4.1: Rezultati po obraznih značilkah 1/3

% vseh obrazov	Zunanji rob leve obrvi	Notranji rob leve obrvi	Zunanji rob desne obrvi	Notranji rob desne obrve
10	0,0526569	0,0794129	0,0407637	0,0793576
20	0,0666302	0,091721	0,0566152	0,0911372
30	0,0773168	0,1037434	0,0681628	0,1009302
40	0,087768	0,1115822	0,079485	0,1106016
50	0,096861	0,1226085	0,090525	0,1209165
60	0,1076606	0,1347452	0,1021122	0,128534
70	0,1168838	0,1485294	0,1140535	0,1387559
80	0,1305086	0,1674022	0,1278854	0,1511286
90	0,1527388	0,189739	0,1504004	0,1718279
100	4,442919	3,980476	4,134149	4,020373

Tabela 4.2: Rezultati po obraznih značilkah 2/3

% vseh obrazov	Zunanji rob levega očesa	Notranji rob levega očesa	Zunanji rob desnega očesa	Notranji rob desnega očesa
10	0,0308803	0,0192355	0,0252198	0,0322333
20	0,038509	0,0258542	0,031067	0,0426386
30	0,0444484	0,0298931	0,0356713	0,048766
40	0,0493314	0,034097	0,040017	0,05394
50	0,0532665	0,038435	0,0435725	0,059028
60	0,0572144	0,0426586	0,0478822	0,0637994
70	0,0637612	0,0466143	0,0524922	0,0693356
80	0,0706626	0,0516988	0,058918	0,076286
90	0,0820474	0,0610116	0,0715068	0,0878248
100	4,231257	4,179329	3,917153	3,877348

Tabela 4.3: Rezultati po obraznih značilkah 3/3

4.2 Testiranja hitrosti programskega ogrodja

Testirali smo hitrost detekcije, sledenja in izvedbe celotnega cevovoda vključno z obogatitvami in prikazom. Rezultati testiranja so namenjeni primerjavi hitrosti procesiranja v odvisnosti od vhodnih parametrov. Izmerjene hitrosti so odvisne od hitrosti računalnika, kvalitete kamere in zunanjih parametrov

(na primer osvetlitev prostora ali pozicija kamere). Testirali smo na štiri jedrnem Intelovem $i5$ $2.67GHz$ procesorju s $4GB$ hitrega spomina.

Testiranje hitrosti izvedbe celotnega cevovoda smo izvedli na dveh videih ločljivosti 1280×720 in 640×360 slikovnih točk, vsebina obeh je bila enaka. Za čas izvedbe smo vzeli povprečno vrednost iz sekvence petstotih slik. V tabeli 4.4 so časi meritev izvedbe celotne zanke pri sledenju samo enega obraza. Vidimo lahko, da je najbolj pomemben faktor tukaj ločljivost prikaza. Čeprav imamo pri prikazu na zaslonu ločljivosti 1900×1080 slikovnih točk s 43 milisekundami še vedno predvajanje videa v realnem času.

Zajem	**Detekcija**	**Sledenje**	**Prikaz 1080p**	**Prikaz 720p**
360p	360p	360p	*31 ms*	*31 ms*
720p	360p	360p	*43 ms*	*31 ms*
720p	720p	720p	*43 ms*	*31 ms*

Tabela 4.4: Izmerjenji časi od zajema do prikaza slike na ekranu pri sledenju enega obraza.

Zajem	**Detekcija**	**Sledenje**	**Prikaz 1080p**	**Prikaz 720p**
720p	360p	360p	*60 ms*	*41 ms*
720p	360p	720p	*59 ms*	*41 ms*
720p	720p	720p	*72 ms*	*51 ms*

Tabela 4.5: Izmerjenji časi od zajema do prikaza slike na ekranu pri sledenju štirim obrazom.

Rezultati sledenja štirim osebam so vidni v tabeli 4.5, kjer je zanimivo to, da je skupen čas manjši pri ločljivosti sledenja 720p v primerjavi s 360p. Temu je tako, ker je zmanjšanje ločljivosti slike časovno zahtevna operacija in se zato izniči prednost, ki jo dobimo pri sledenju obrazu na manjši sliki, čeprav smo za interpolacijo izbrali metodo najbližjih sosedov, ki je bila najhitrejša med metodami, ki so implementirane v knjižnici OpenCV.

Tako v najslabšem primeru pri enem obrazu dosežemo hitrost nad 23 okvirjev na sekundo, pri štirih obrazih pa nad 13 okvirjev na sekundo. Za

kontrast v najboljših primerih dosežemo nad 32 oziroma nad 25 okvirov na sekundo. Torej dosegamo delovanje v realnem času.

Poglavje 5

Zaključek

Pri implementaciji programskega ogrodja smo ugotovili, da je večina algoritmov računalniškega vida procesorsko potratnih, kar omejuje funkcionalnost. Zato aplikacije, ki uporabljajo obogateno resničnost za interakcijo z uporabnikom še niso tako razširjene. Drugi problem pri takšnih aplikacijah je velikost prikazovalnika, kot je na primer ekran na pametnem telefonu, ki omejuje uporabnikov pogled na okolje, saj mora gledati v ekran za prikaz dodane grafike. To veliko prepreko rešujejo naprave, ki šele prihajajo na trg, kot so očala s prikazom grafičnega vmesnika na lečah. Naša aplikacija reši problem prikazovalnika s tem da je osnovna funkcionalnost takšna, da uporabnik gleda sebe na velikem ekranu in s tem dobi ekran funkcijo ogledala.

Za detekcijo obrazov smo uporabili algoritem Viola-Jones, ki zadovoljivo hitro detektira obraze, vendar je prepočasen za detekcijo v realnem času. Za oceno oddaljenosti od kamere, smo uporabili preprost algoritem, ki temelji na modelu točkaste kamere in daje dovolj dobre rezultate za praktično uporabo. Veliko večji je bil problem pri iskanju algoritma za iskanje obraznih značilk, saj smo potrebovali robusten in predvsem točen algoritem. S točnostjo dosežemo to, da so objekti obogatene resničnosti narisani čim bolj realno na obraz. S tem mislimo na to, da je pozicija, velikost in rotacija takšna, da objekt izgleda čim bolj naravno na obrazu. Za detekcijo značilk smo uporabili izboljšano verzijo algoritma aktivnega modela videza. Glede

na rezultate testiranja, ki smo ga opravili v četrtem poglavju lahko rečemo, da je detekcija obraznih značilk dovolj dobra za praktično uporabo za našo aplikacijo.

Aplikacija uporabnikom omogoča hitro konfiguracijo parametrov preko preprostega vmesnika. Prav tako ogrodje omogoča programerjem hitro in preprosto integracijo z njihovo aplikacijo preko programskega vmesnika, ki je opisan v dodatku A ter omogoča enostavno dodajanje novih modulov.

Glede na vsa zapisana dejstva nam je uspelo implementirati programsko ogrodje, ki ustreza vsem pogojem, ki smo si jih zastavili.

5.1 Možne izboljšave

Ker je bila pri procesiranju zelo pomembna zahteva delovanje v realnem času, smo tudi pri implementaciji programskega ogrodja naredili kompromis med funkcionalnostjo in hitrostjo. To pomeni da smo omejili funkcionalnost tako, da je procesiranje možno v realnem času. Na primer pri problemu sledenja obrazu smo poleg algoritma za detekcijo obraznih značilk, potrebovali tudi algoritem za sledenje značilkam, saj je detekcija računsko preveč zahtevna, da bi lahko sprocesirala vsako sliko posebej iz slikovnega toka.

Obstajajo seveda rešitve, ki lahko optimizirajo izvedbo. Na primer v knjižnici OpenCV imamo implementiran modul za procesiranje na grafični enoti. Ta je še posebej primerna za vzporedno procesiranje, kar lahko dobro izkoristimo pri algoritmih za računalniški vid. Vendar bi s tem omejili funkcionalnost, saj je implementacija odvisna od proizvajalca grafične kartice. Tako bi programsko ogrodje delalo samo na računalnikih, ki imajo določeno znamko grafične kartice. Sicer je v zadnji verziji knjižnice OpenCV 3.0 modul za procesiranje na grafični enoti narejen tako, da je za programerja vseeno, kakšna je znamka grafične kartice. Ampak ta knjižnica med razvojem programskega ogrodja še ni bila na voljo, saj je izšla šele mesec nazaj. Dodan ima tudi modul, ki sledi objekte v realnem času z uporabo algoritmov za strojno učenje in se tako sproti uči, kateri objekt naj sledi. S tem prido-

bimo na robustnosti sledenja, saj bi lahko obrazu še naprej sledili tudi, če se uporabnik obrne in nimamo direktnega pogleda na obraz.

Dodatne funkcionalnosti bi lahko bile na primer sledenje telesu, ocena poze telesa, zaznavanje kretenj z rokami, razpoznava objektov ipd. Določeni algoritmi za ta namen so že implementirani v knjižnici OpenCV in bi jih zato lahko brez težav uporabili v programskem ogrodju.

Vendar pa lahko postane problem procesiranja vseh teh informacij v realnem času. Smiselna rešitev takšnega problema bi bila lahko procesiranje v oblaku, vendar bi potrebovali hitro internetno povezavo strežnika z napravo.

Dodatek A

Opis programskega vmesnika za razvoj aplikacij na osnovi programskega ogrodja

V tem poglavju bomo opisali naš programski vmesnik (angl. Application Programming Interface). Za upravljanje s programskim ogrodjem smo definirali razred FrameworkAPI (slika A.1). Ostale nastavitve, zaradi bolj enostavne uporabe, nastavimo v XML datoteki. Sledi opis metod vmesnika in razlaga XML strukture:

```
void startFramework(int msec=45)
```

Vhod: Na vhodu definiramo minimalni časovni interval, v katerem zajamemo sliko iz slikovnega toka. Če procesiranje zanke traja dlje, kot je naveden, čas potem zajame naslednjo sliko takrat, ko se glavna zanka zaključi. Privzeta naprava za zajem se definira v razredu `Const`.

Procesiranje: Funkcija inicializira glavno zanko programskega ogrodja za procesiranje videa, ki je implementirana z nitjo, tako glavna zanka procesira paralelno.

Izhod: Ko se glavna zanka konča v vzporednem procesu, se signalizira

metodi `frameReady(cv::Mat)` in kot parameter pošlje sliko. Programsko ogrodje sliko nato pretvori v format `QImage` in s signalom `onProcessFinished(QImage)` signalizira, da je procesiranje slike končano. Programer ta signal poveže z ustrezno funkcijo za prikaz slikovnega toka na grafičnem vmesniku.

`void startFramework(std::string, int msec=45)`

Vhod: Funkcija na vhodu poleg intervala zajemanja slike dobi tudi naslov video datoteke.

Procesiranje in izhod: Funkcionalnost je ista kot v prejšnji metodi.

`cv::Mat processImage(cv::Mat)`

Vhod: Ime slikovne datoteke.

Procesiranje: Funkcija je namenjena za procesiranje slike.

Izhod: Vrne procesirano sliko.

`vector<` $cv :: Rect$ `> detectFace (cv::Mat frame)`

Vhod: Slika.

Procesiranje: Detektira vse obraze na sliki.

Izhod: Funkcija vrne seznam področij, kjer je detektirala obraze.

`float estimateDistance(cv::Mat)`

Vhod: Slika obraza.

Procesiranje: Oceni oddaljenost obraza od kamere.

Izhod: Vrne oddaljenost obraza v centimetrih.

`vector<float> estimateFacePose (cv::Mat)`

Vhod: Slika obraza.

Procesiranje: Oceni pozo obraza v prostoru glede na kamero.

Izhod: Vrne tri vrednosti kotov. Prva vrednost je kot glede na horizontalno os. Druga vrednost je kot glede na vertikalno os. Tretja vrednost je kot glede na os, ki povezuje obraz in kamero.

`cv::Mat getFaceLandmarks(cv::Mat)`

Vhod: Slika obraza.

Procesiranje: Oceni pozicijo obraznih značilk.

Izhod: Vrne x in y pozicijo 49-ih obraznih značilk v tabeli tipa `cv::Mat`.

`bool useDisplay()`

Vhod: Nima vhodnih podatkov.

Procesiranje: Funkcija pove ali se uporablja standardni izhod ali ne.

Izhod: Vrne *true*, če se uporablja standardni izhod, nasprotno vrne *false*.

`void setUseDisplay(bool)`

Vhod: Na vhodu potrebuje logično vrednost *true* za uporabo standardnega izhoda, nasprotno vrednost *false*.

Procesiranje: Funkcija nastavi uporabo standardnega izhoda.

Izhod: Funkcija ne vrne nobene vrednosti.

`void setDisplayResolution (int width, int height)`

Vhod: Na vhodu potrebuje vrednost za širino in višino prikazane slike.

Procesiranje: Funkcija nastavi ločljivost izhodne slike. Ta vrednost se lahko nastavi v XML datoteki. Metoda je implementirana zato, da se lahko ločljivost prikaza spreminja dinamično med delovanjem programskega ogrodja.

Izhod: Funkcija ne vrne nobene vrednosti. Vendar pa s signalom `displResolutionSignal(QPoint)` sporoči, da se je spremenila

ločljivost izhodne slike. Programer potem poskrbi, da se grafični vmesnik primerno poveča ali pomanjša.

`void setCaptureResolution (int width, int height)`

Vhod: Na vhodu potrebuje vrednost za širino in višino slike.

Procesiranje: Funkcija nastavi ločljivost vhodne slike. Ta vrednost se prav tako lahko nastavi v XML datoteki.

Izhod: Funkcija ne vrne nobene vrednosti.

`QPoint CaptureResolution ()`

Vhod: Nima vhodnih podatkov.

Procesiranje: Prebere ločljivost zajemanja slike.

Izhod: Funkcija vrne ločljivost v formatu `QPoint`.

`QPoint DisplayResolution ()`

Vhod: Nima vhodnih podatkov.

Procesiranje: Prebere ločljivost prikazane slike.

Izhod: Vrne ločljivost v formatu `QPoint` .

Ostale parametre, zaradi preprostosti, nastavimo v XML datoteki:

`<facesGroup>` Definira skupino ljudi. Skupin je lahko poljubno veliko.

`<distanceFrom>, <distanceTo>` Interval oddaljenosti obraza od kamere.

`<faceItem>` Razred definira sliko, ki se bo narisala na obraz.

`<class>` Pozicija na obrazu. Možne vrednost: `CenterFace, Chin, Forehead, Nose, LeftEyebrow, RightEyebrow, RightEye, LeftEye, RightCheek, LeftCheek, Mustache.`

`<image>` Ime datoteke slike.

`<scale>` Faktor velikosti slike glede na širino obraza.

`<offset.x>, <offset.y>` Oddaljenost od obrazne značilke.

`<MindreaderItem>` Razred definira stripovski oblaček nad obrazom.

`<R>, <G>, <B>` Vrednosti barve v RGB prostoru.

`<transparency>` Prozornost oblačka. Vrednost mora biti v intervalu [0..1].

`<filename>` Ime tekstovne datoteke, kjer je shranjen scenarij. Vsebuje vsebino, ki se naključno prikaže v oblačku.

`<settings>` Shrani vse nastavitve, ki se lahko nastavijo v uporabniškem vmesniku.

`<captureResolution>` Ločljivost zajetja slike iz kamere.

`<detectionResolution>` Ločljivost slike, ki jo uporablja detektor.

`<trackingResolution>` Ločljivost slike, ki jo uporablja modul za sledenje obrazom.

`<numberOfFacesTracked>` Maksimalno število obrazov, ki se jim sledi.

`<rotationIndex>` Sliko lahko zarotiramo v desno ali levo za 90 stopinj. Vrednost -1 pomeni rotacijo v levo za 90 stopinj, 1 pomeni rotacijo v desno za 90 stopinj.

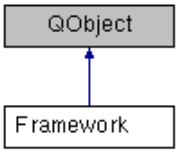
`<useDebug>` Če je vrednost nastavljena na 1, prikaže dodatne informacije na sliki za potrebe razhroščevanja, nasprotno je vrednost 0.

`<ROI>` Polje omeji področje detekcije obrazov na sliki.

`<banner>` Definira reklamno pasico in pozicijo na sliki.

FrameworkAPI Class Reference

Inheritance diagram for FrameworkAPI:



Public Slots

void	**process** ()
void	**frameReady** (cv::Mat)

Signals

void	**onProcessFinished** (QImage)
void	**displResolutionSignal** (QPoint)

Public Member Functions

void	**startFramework** (int msec=45)
void	**startFramework** (std::string, int msec=45)
cv::Mat	**processImage** (cv::Mat)
vector< cv::Rect >	**detectFace** (cv::Mat frame)
void	**setDisplayResolution** (int width, int height)
QPoint	**displayResolution** ()
void	**setCaptureResolution** (int width, int height)
QPoint	**captureResolution** ()
bool	**useDisplay** ()
void	**setUseDisplay** (bool)
cv::Mat	**getFaceLandmarks** (cv::Mat)
float	**estimateDistance** (cv::Mat)
vector< float >	**estimateFacePose** (cv::Mat)

Slika A.1: Razred FrameworkAPI.

Slike

Tabele

Literatura

[1] R. T. Azuma et al. A survey of augmented reality. *Presence*, 6(4):355-385, 1997.

[2] C. Baral, V. Kreinovich, R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1):241-267, 2000.

[3] P. N. Belhumeur, D. W. Jacobs, D. Kriegman, N. Kumar. Localizing parts of faces using a consensus of exemplars. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, str. 545-552, IEEE, 2011.

[4] J. Carmigniani, B. Furht. Augmented reality: an overview. *Handbook of augmented reality*, str. 3-46, Springer, 2011.

[5] T. Cootes, E. Baldock, J. Graham. An introduction to active shape models. *Image Processing and Analysis*, str. 223-248, Oxford University Press, 2000.

[6] D. Cristinacce, T. Cootes. Feature detection and tracking with constrained local models. *British Machine Vision Conference (BMV)*, str. 929-938, 2006.

[7] D. F. Dementhon, L. S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1-2):123-141, 1995.

[8] B. Efraty, C. Huang, S. Shah, I. A. Kakadiaris. Facial landmark detection in uncontrolled conditions. *International Joint Conference on Biometrics (IJCB)*, str. 1-8, IEEE, 2011.

[9] X. Gao, Y. Su, X. Li, D. Tao. A review of active appearance models. *IEEE Transactions on Systems, Man, Cybernetics, Part C: Applications and Reviews*, 40(2):145-158, 2010.

[10] L. Hou, X. Wang, L. Bernold, P. Love. Using animated augmented reality to cognitively guide assembly. *Journal of Computing in Civil Engineering*, 27(5):439-451, 2013.

[11] Z. Huang, F. Ren. Facial expression recognition based on active appearance model and scale-invariant feature transform. *IEEE International Symposium on System Integration (SII)*, str. 94-99, IEEE, 2013.

[12] O. Jesorsky, K. J. Kirchberg, R. W. Frischholz. Robust face detection using the hausdor distance. *Audio and video based biometric person authentication*, str. 90-95, Springer, 2001.

[13] G. A. Lee, A. Dunser, S. Kim, M. Billinghurst. CityViewAR: A mobile outdoor AR application for city visualization. *IEEE International Symposium on Mixed and Augmented Reality*, str. 57-64, IEEE, 2012.

[14] R. Lienhart, J. Maydt. An extended set of haar-like features for rapid object detection. *IEEE International Conference on Image Processing (ICIP)*, str. 900-903, IEEE, 2002.

[15] D. Litwiller. CCD vs. CMOS. *Photonics Spectra*, 35(1):154-158, 2001.

[16] D. G. Lowe. Object recognition from local scale-invariant features. *IEEE International Conference, Volume 2*, str. 1150-1157, IEEE, 1999.

[17] I. Matthews, S. Baker. Active appearance models revisited. *International Journal of Computer Vision*, 60(2):135-164, 2004.

[18] S. Milborrow, F. Nicolls. Locating facial features with an extended active shape model. *ECCV Computer Vision*, str. 504-513, Springer, 2008.

[19] P. Milgram, H. Takemura, A. Utsumi, F. Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. *Photonics for Industrial Applications*, str. 282-292, International Society for Optics and Photonics, 1995.

[20] E. Murphy-Chutorian, M. M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):607-626, 2009.

[21] T. Musek. Merjenje osredotočenosti in razdalje na osnovi detekcije obraza. *Tehnično poročilo*, Fakulteta za računalništvo in informatiko UL, 2014.

[22] D. Roberts, A. Menozzi, J. Cook, T. Sherrill, S. Snarski. Testing and evaluation of a wearable augmented reality system for natural outdoor environments. *SPIE Defense and Security and Sensing*, str. 87350A-87350A, SPIE, 2013.

[23] T. Sielhorst, M. Feuerstein, N. Navab. Advanced medical displays: A literature review of augmented reality. *Journal of Display Technology*, 4(4):451-467, 2008.

[24] I. E. Sutherland. A head-mounted three dimensional display. *AFIPS Fall Joint Computer Conference, part I*, str. 757-764, ACM, 1968.

[25] M. Valstar, B. Martinez, X. Binefa, M. Pantic. Facial point detection using boosted regression and graph models. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, str. 2729-2736, IEEE, 2010.

[26] P. Viola, M. Jones. Rapid object detection using a boosted cascade of simple features. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),volume 1*, str. 501-511, IEEE, 2001.

[27] X. Xiong, F. De la Torre. Supervised descent method and its applications to face alignment. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 532-539, IEEE, 2013.

[28] S. Yuen, G. Yaoyuneyong, E. Johnson. Augmented reality: An overview and five directions for AR in education. *Journal of Educational Technology Development and Exchange*, 4(1):119, 2011.

[29] (2014) Atheer Labs. Dostopno na:
https://www.atheerlabs.com

[30] (2014) AUDI eKurzinfo. Dostopno na:
http://www.metaio.com/customers/case-studies/audi-ekurzinfo- app

[31] (2014) Global Augmented Reality (AR) Market Forecast 2011. Dostopno na:
http://www.marketsandmarkets.com/PressReleases/augmented-reality-virtual-reality.asp

[32] (2014) Nissan RTT. Dostopno na:
http://www.rtt.ag/en

[33] (2014) Toyota 86 AR. Dostopno na:
http://www.toyota86ar.com

[34] (2014) IKEA Catalogue Android App. Dostopno na:
play.google.com/store/apps/details?id=com.ikea.catalogue.android

[35] (2014) Lego 3D. Dostopno na:
https://www.vuforia.com/case- studies/lego-connect

[36] (2014) Google glasses. Dostopno na:
https://www.google.com/glass/start

[37] (2014) Space Glasses. Dostopno na:
https://www.spaceglasses.com

[38] (2014) Tele-pathy.org. Dostopno na:
http://tele-pathy.org

[39] (2014) OpenCV. Dostopno na:
http://www.opencv.org

[40] (2014) CCV: A Modern Computer Vision Library. Dostopno na:
http://libccv.org

[41] (2014) Simple CV. Dostopno na:
http://simplecv.org

[42] (2014) PAMI Mark Everingham Prize. Dostopno na:
http://www.computer.org/portal/web/tcpami/Everingham-Prize

[43] (2014) GIT. Dostopno na:
http://git-scm.com

[44] (2014) Extensible Markup Language (XML) 1.0. Dostopno na:
http://www.w3.org/XML

[45] (2014) SQLite. Dostopno na:
http://www.sqlite.org

[46] (2014) QCustomPlot. Dostopno na:
http://www.qcustomplot.com

[47] (2014)Visual Studio IDE. Dostopno na:
http://msdn.microsoft.com/en-us/library/dd831853.aspx